Intro to Linux

In this guide I'm gonna try and give you the essential information you need to be able to navigate the Unix command-line. I'll try to make it as short and sweet as possible, but *take your time to make sure you really understand everything*. That's the whole point. Note that I am **not** covering how to SSH into the UTCS Unix machines here (i.e., log in remotely from your own computer); that's a whole 'nother topic. Check out the other tutorial on that!

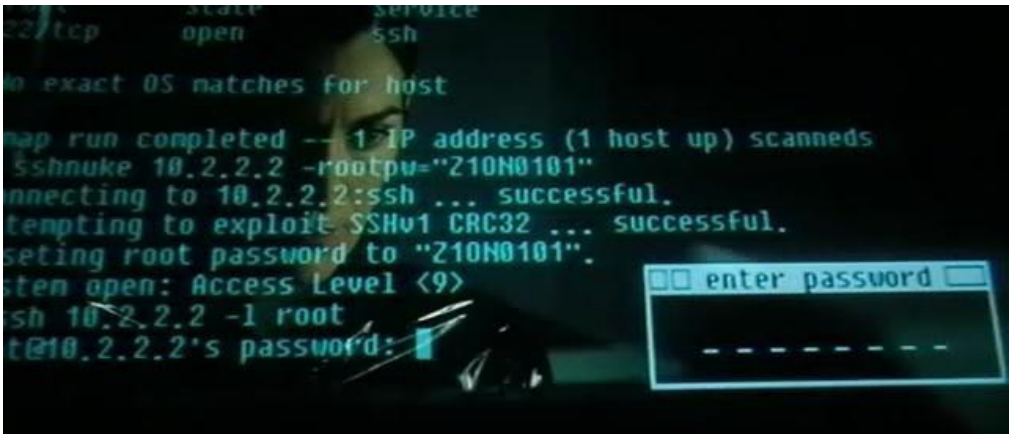
If you haven't gotten a UTCS Unix account for logging into the machines, create one now. Walk over to one of the labs in GDC. Log in.

UNIX/LINUX

You've probably heard of Unix, right? The Unix operating system was developed by of a group of AT&T employees in 1969. It's proprietary. Copyrighted. Big companies use it. You've probably also heard of Linux, but do you know the difference between the two? Linux is a Unix clone, written from scratch by Linus Torvalds, and is free and open source!
Don't get confused though - "Linux" is actually just a kernel, an important *piece* of an operating system but useless by itself. A Linux *distribution* is the whole package -- it includes the Linux kernel as well as various other applications necessary to form a complete usable operating system (GUI system, GNU utilities, compilers, editors, etc.). In reality most people use the term "Linux" to refer the whole operating system, but there you have it. Now you're all the wiser! One of the most popular Linux distributions out there is Ubuntu and is what you'll find on the UTCS machines.

WHAT IS THE TERMINAL/SHELL/CONSOLE/COMMAND LIASJK31LD;KSAJDM...?

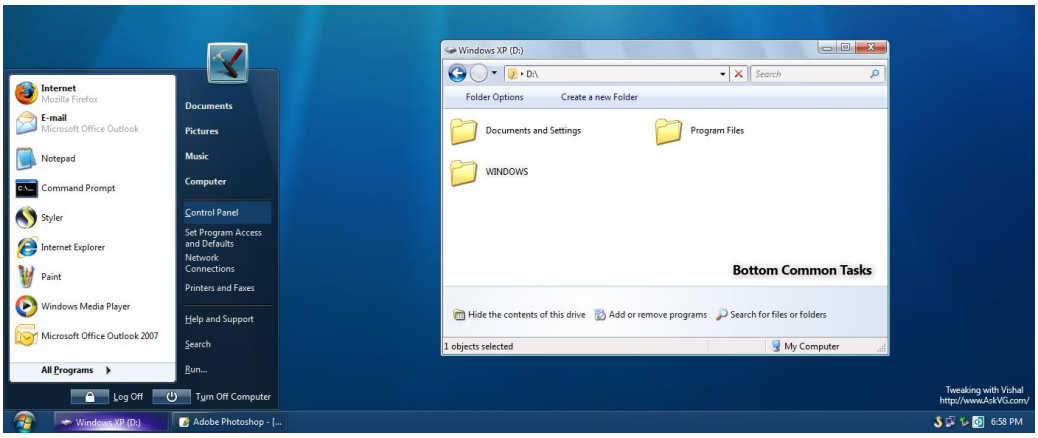
If you've ever come across a hacking scene in a movie, you probably saw a pair of hands typing at lightning speed and a stream of undecipherable text flooding the screen, partially masked by the concentrated expression of the hacker's reflection. But obviously, only expert hackers work like this.



Hack that power grid, Trinity!

What? You want to do that too? Great! Once you complete this guide and learn how to properly use the command-line, you'll be exactly like that!*

You're probably used to working on a computer through the world of GUIs (Graphical User Interface), am I right? Everything you need has a nice *graphic*, button, or icon that you can click to tell the computer what you want it to do.



Say goodbye to pretty interfaces

Well once upon a time those things didn't exist. Once upon a time you had to type *text* commands to tell the computer what you wanted it to do. Oh please, it's not as bad as it sounds.

*Note: *You will not be exactly like that.*

Now if you log onto one of the UTCS Linux machines right now, there *is* a GUI system (breathe a sigh of relief), and you'll find that you can pretty much get away with not using the command line for many tasks such as creating directories, deleting files, etc. But a programmer that knows their way around the shell could do something slightly more complex like, say... delete all files on your computer with names that begin with the phrase "`hubaloo`", all with one command. How fast could you do that by clicking your mouse? '*Not fast'* is the right answer. Once you bring up a terminal, the whole point is that you shouldn't need to use your mouse anymore.

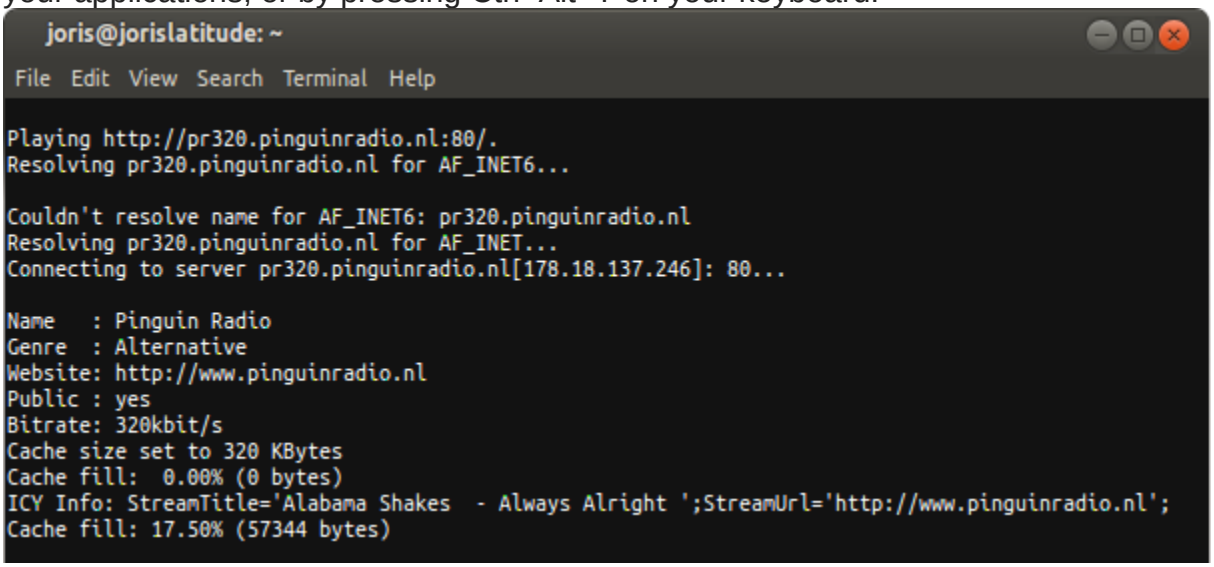
So let's get started.

Firstly, 'terminal', 'shell', 'command-line', 'console'... which term is correct? There *is* technically a difference between them, but I will use the names interchangeably because... I don't really care, and for the sake of this guide it's not important!

OK. How do I start the shell?

IF YOU'RE ON LINUX:

Just so you know, the default shell on Ubuntu is called the Bourne Again Shell (or "bash"). It's what you'll be using. You can start a session by searching for "Terminal" in your applications, or by pressing Ctrl+Alt+T on your keyboard:



An example of the Bash Unix shell

IF YOU'RE ON A MAC:

Besides making your wallet feel a lot lighter, Macs also provide you with a Unix-like operating system. That's good news! Apple's OS X is a close relative of FreeBSD, a UNIX descendant. What does that mean for you? For one, if you ever decide to pull up your terminal on a Mac, you'll find that the command line is virtually the same. So if developing from home is your thing, your life just got a lot easier. Open the 'Terminal' Application that came with your Mac, or download a better one.

Nearly all of what is in this guide can be applied to a Mac as well, so tomato-tomahto. You can also SSH into a Unix machine if you like. However, I still recommend hiking over to the UTCS lab for a more *genuine* Linux experience.

IF YOU'RE ON WINDOWS:

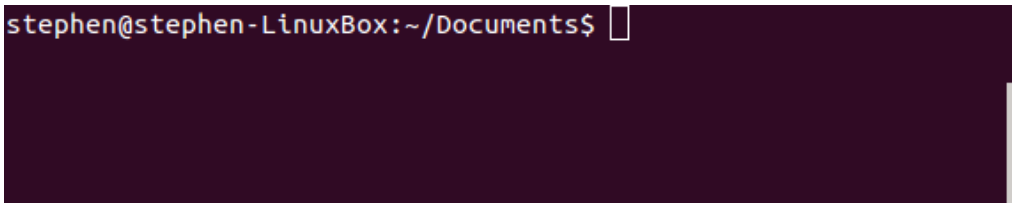
Sorry, bud. Windows has gone down their own path for a while, so this guide doesn't really apply to the Windows command line. But do not despair! There are a plethora of programs out there that can help you achieve interaction with the UTCS machines the same way any other Unix-like OS would.

Firstly, I recommend getting your butt over to GDC and getting on an actual Linux machine to practice. Or use a Windows SSH client, such as PuTTY, to log into a UTCS machine remotely (check the other SSH guide for that). I recommend the former for newcomers.

Alternatively, to get Linux capabilities on your own computer you may also want to check out Cygwin, dual booting, and virtual machines.

THE COMMAND LINE

The first thing you should see is your command prompt. It is usually a sequence of characters such as **$** or **>** and often includes the username, host name, and current working directory. So if you see those characters in one of my examples, do NOT actually type the prompt; it is not a part of the command.

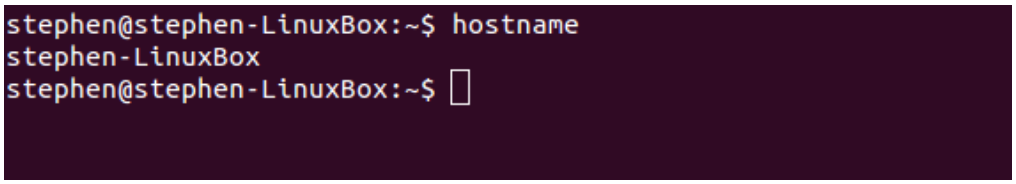


An example prompt. This prompt displays the user (`stephen`), the host name (`stephen-LinuxBox`), the current working directory (`~/Documents`), and the prompt character `$`

The prompt is the computer's way of telling you it is ready to accept commands. Each time you type a command and hit Enter, the machine will execute that command, show any output, and then give you another prompt.
Go ahead and hit Enter a couple times. You will see nothing happens, and another prompt appears. You are basically telling the computer to execute *no* command.

`hostname` – prints the name of the machine

Try typing `hostname` and hitting Enter. You should see the name of the machine you are currently on. You have just told the computer to find out its name and print it to the terminal. You're on your way!



The `hostname` command displays the name of the machine

Most importantly, don't be scared by the command-line! Misspelling a command or typing a wrong option will (probably) not crash your computer or anything. You'll most likely just see "`command not found`" or some advice on how to fix it.

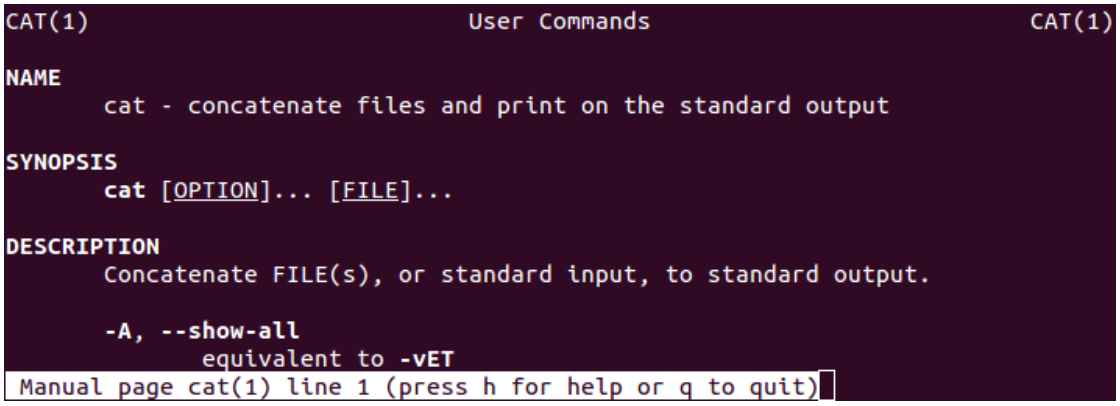
EARLY ADVICE: Make Your Life Much Easier

Man pages: One of the **best** resources on learning about a command is the manual pages. You can pull up a "man page" by typing `man` and then the name of the command you're interested in.

> `man` – brings up the manual page for a command. This page offers a description of the command, what options and arguments are available, example uses, etc. Options/arguments within [brackets] are optional, ones followed by … mean multiple can be used, and all others are required.

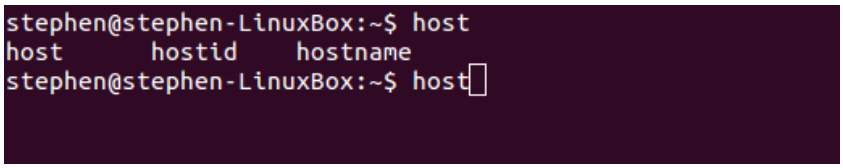
Navigate a man page with the up and down arrow keys, and press Q to quit. Try learning about the command we used earlier by entering:

`man hostname`.



`Cat man` produces the manual page for the `cat` command

Tab autocomplete: Take full advantage of the autocomplete feature. If you forgot how to spell the second half of a command name, file name, directory, or option name (or you are just too lazy to write it), hit Tab to have the shell complete it for you! If there are multiple possible options that the shell can choose from, hit Tab twice to see all the autocompletion candidates. Try typing an incomplete command name like `host` and then hitting Tab twice to see some options. Add an `n` to make `hostn` and hit Tab; as you can see, the rest of the `hostname` command is automatically filled in for you.



Hitting Tab twice after typing "`host`" displays candidates for completion. The terminal will guess what you meant to write. Works for commands, files, and directories as well.

Using previous commands: Use up and down arrow keys on the command line to navigate through your command history. This is a great way to quickly pull up a complex command you wrote earlier.

Clear your screen: When things are getting cluttered, use the `clear` command to clear the terminal screen.

```
clear – clear the terminal screen
```

Terminating a process: As a last resort, if you make an oopsie and need to terminate a command or exit the running program, hit CTRL+C. This will send a signal to kill the current process.

STRUCTURE OF COMMANDS: How to Write 'em

You can type a command name, but often times you need to provide more information about how to execute that command or data to execute it on. Which file do you delete? Which directory does this go in? What phrase do you search for? This extra information comes in the form of *options* and *arguments*, after the command name. Here is the basic structure:

```
command [-option(s)] [argument(s)]
```

Options modify how the command executes; they are usually in the form of a dash followed by a letter in the alphabet (sometimes two dashes followed by a word). *Arguments* usually identify the data upon which the command performs. Consider the following command, which searches the file named "`essay.txt`" for the phrase '`searchMe`':

```
grep -i searchMe essay.txt
```

It's okay if you're not familiar with the `grep` command. Let's break this down…

- **Command name:** This is the `grep` command, which is used to search files for a pattern or phrase.
- **Options:** We used the `-i` option, which alters the functionality of `grep` so that all searching is case-insensitive. This will now accept phrases such as `SEARCHME`, `sEaRcHmE`, `searchme`, etc. If we had excluded this option, grep would have been picky and only the exact case of `searchMe` would have been acceptable.
- **Arguments:** The `grep` command takes several arguments: the first is the pattern you are searching for (in this case "`searchMe`"). Then you can list the names of any number of files you want to search through (in this case only `essay.txt`). Now the computer knows which file to open up and what phrase to search for.

Not so bad, right? Remember: command, options, arguments. Let's look at a few other examples of basic Linux commands you may run across (*options* are blue, *arguments* are red):

```
ls -la
tar -xvzf myFileArchive.tar.gz
diff file1.txt file2.txt
rm -rf badDirectory
```

Sometimes commands can get humongous and confusing. If you ever want to break down what a command is doing, try this nifty site that explains commands piece-by-piece: www.explainshell.com

NAVIGATING THE FILE SYSTEM: Getting Around

- **Objective:** Learn to find your way around files and directories with the `pwd`, `ls`, and `cd` commands. Understand the meaning of `~`, `.` and `..`
- **Commands:** `pwd, echo, ls, cd`

If you haven't already, download your very own "Linux Starter Kit" tarball to your Downloads folder with the following command:

```
wget -P ~/Downloads/ https://webspace.utexas.edu/stc493/LinuxTutorials/linux-starter-kit.tar
```

Tar files are commonly used in Unix systems to combine and compress multiple files, like RAR or ZIP files. We will be using the files in this kit for many of our exercises. Let's find out how we can navigate over to the Downloads folder.

First let's get our bearings. Where are we currently? What directory are we in? Sounds a job for `pwd`!

> `pwd` – stands for "print working directory"; prints the full name of the directory you are currently in

Try it. You should see something like `/u/username`. If not, enter this command:

```
cd ~
```

This is your home directory, where you begin when you first start the shell. Your home directory can also be referenced with the tilde `~` symbol. Let's illustrate that with the `echo` command.

> `echo` – prints the argument right back at you into the terminal

Try echoing "hello world":

```
echo hello world
```

As you can see, it just echoed your argument right back to you. Now echo the tilde ~:

```
echo ~
```

Hey, just like we mentioned, your terminal interprets the tilde `~` as your home directory `/u/username`. This will come in handy later!

Time to see what files are in our current directory.

> `ls` – lists the contents of the current directory

If you enter `ls`, you should see a list of the typical files and directories would expect to see: Documents, Downloads, Pictures, etc. Now add some options to the command:

```
ls -la
```

The `-l` option formats the contents in a nice list, and the `-a` option will print *all* files. Wait, you say, now I see these files with periods like `.` and `..`? Well those are special. The single period means "this directory", and the double period means "the parent directory" (the directory above this one). Of course, these values will change to adapt to whichever directory you are in.

Let's start moving around. You can change your current directory with the `cd` command. Finally we're getting somewhere…

`cd` – "change directory"; go to the directory supplied in the argument

Use `cd` to enter the Documents directory:

```
cd Documents
```

You're now in the Documents folder. Try some `pwd` and `ls` commands to see what's in there.
How do you get back out? Remember, the double-periods referred to the parent directory. So you can just enter:

```
cd ..
```

Also remember, the tilde ~ refers to the home directory. So you could also do the following to go back to your home dir:

```
cd ~/
```

You can create very complex paths to files or directories. A *relative* path is based on where you currently are, such as:

```
../../../Downloads/projects/
```

The above path means *go three parents up from the current directory, then into the `Downloads` folder, then into the `projects` folder*.

An *absolute* ("fully-qualified") path gives the entire path from the root all the way down to the directory or file, such as:

```
u/stc/Documents/foo/hello.txt
```

As you can imagine, absolute path names can be long if a file is way deep in the file hierarchy. Remember to use Tab completion to help you write paths quicker!

Using this knowledge, let's open up that Linux Starter Kit you downloaded earlier. Go over to the Downloads folder by using the `cd` command:

```
cd ~/Downloads
```

You should find the Linux Starter Kit tar file you downloaded earlier in this directory. Open up the tar file.

```
tar xvf linux-starter-kit.tar
```

This will extract the files in the tarball. If you `ls` again, you should now see a `linux-starter-kit` directory. Enter the directory using `cd`:

```
cd linux-starter-kit
```

Shall we begin?

MANIPULATING FILES: Mess with Stuff

- **Objective:** Learn everyday file operations: copy, move, delete, read, create file, create directory, turn in a UTCS assignment
- **Commands**: `cp, mv, rm, rmdir, cat, less, touch, mkdir, turnin`

We're inside the starter kit directory. First, do an `ls` to see what's in here. You should see a couple files, as well as the `maze` and `playground` directories – we'll get to those later.

Perhaps you already saw a tantalizing file called `readMe.txt`. What's in there?! What's the best way to get a taste of its contents?

> `cat` – short for "concatenate"; a versatile command for combining files, creating new files, or printing the contents to the screen

We'll use the `cat` command initially to print out files. Try calling this command on the readme file:

```
cat readMe.txt
```

It should spit the contents of the file right to the terminal. Quick and easy. But let's say it was a really really long file and you would rather not have all the contents thrown onto the screen at once. Is there a command that lets you interactively scroll through the contents? Yes.

> `less` – displays the contents of an input file, which you can scroll through interactively. Press Q to exit.

Try using `less` on the longer readme file:

```
less longReadMe
```

You're in reading mode now, scroll up or down within the text. Hit Q to exit when you're done reading. Now let's move some files around.

See `copy-me.txt`? Let's make a copy of that file and name it `copy-cat.txt`.

> `cp` – used to copy and/or rename a file from source to a destination

The first argument to `cp` is the source file to copy, and the second argument is the destination file. The corresponding command would be:

```
cp copy-me.txt copy-cat.txt
```

Moving a file, as opposed to copying, is done the same way.

`mv` – used to move and/or rename a file from source to a destination

The `mv` command can also be used just to rename a file. If you move a file to the same location that it was originally in, but with a different name, you have effectively done that, right? Rename `copy-me.txt` to `renamed.txt`:

```
mv copy-me.txt renamed.txt
```

But what if we wanted to move the `copy-cat.txt` file into your Documents directory? You'll have to include a relative or an absolute path for your destination file:

```
mv copy-cat.txt ../../Documents/copy-cat.txt
mv copy-cat.txt ~/Documents/copy-cat.txt
```

After you execute this command, the copy-cat file is no longer in the current directory. If you try an `ls` of the Documents directory, you'll find it enjoying itself over there.

```
ls ~/Documents
```

Okay enough! You want the power of creation and destruction, right? It's easier than you think with files.

`touch` – create a blank file; or update the access time of an existing file

Type `touch` and then a name to create a blank file with that name. Let's create a file with the name badFile:

```
touch badFile
```

Use `ls` and you should see the `badFile` appear in the directory. Right now it's just an empty file. Let's repeat this process, but this time with a directory.

`mkdir` – short for "make directory"; creates a new directory

Make a new directory called `badDir`:

```
mkdir badDir
```

So now we have a "bad" file and "bad" directory. Pretend they're old files, corrupted, or you just want to get rid of them. Now before we go any further, be aware that once you delete a file on your Linux system they will effectively be gone *forever*. If you bug someone you may be able to recover something from the UTCS system snapshots, but the better route is just to be careful!

`rm` – used to delete files. **WARNING**: USE WITH EXTREME CAUTION.

There, the ultimate power of destruction in your hands. First I want to introduce to you an important option: `-i`. The `-i` option will ask you to confirm your action before it

removes the file, a great safeguard for beginners! Delete your `badFile` with this command:

```
rm -i badFile
```

Enter `y` to confirm the removal and poof! It's gone for good.

`rmdir` – short for "remove directory"; deletes an empty directory

Delete the bad directory too (notice there is no `-i` for this command):

```
rmdir badDir
```

Pretty straightforward, yeah? However, notice that `rmdir` only works for *empty* directories. What if you felt like removing the `playground` directory in the Linux Kit? If you check it out you'll see that `playground` already contains several files and directories inside. If you attempt to use `rmdir` on `playground`…

```
rmdir playground
```

You will receive a warning that the directory is not empty, and the command will fail. So how do you delete a non-empty directory and all of its contents?

Be ***very very careful*** with this one, young padawan. You will use the `rm` command with the `-r` option. The `-r` option means it will recursively go through the directory, deleting all files *and* applying the same removal function to its subdirectories, and then those subdirectories' subdirectories, and then those subdirectories' subdirectories… and so on.

Everything from that directory down will be deleted, so be wise where you use this command. (Ex. Calling this from your home or root directory would be a **massive mistake**; you would remove everything you have on the machine.) Now let's remove the playground directory.

```
rm -r playground
```

SEARCHING

- **Objective:** Learn how to search through the contents of files and find certain file names.
- **Commands**: `grep, find`

There's a file out there with a specific text phrase inside. How do you find it? You saw the `grep` command earlier.

> `grep` – searches input files for patterns, and prints the matching lines

There are many ways to use `grep`; I'll cover the basic ones here. Give `grep` a pattern to search for and a file to search through. Let's search the `longReadMe` file from earlier for the pattern "love":

```
grep love longReadMe
```

The terminal will display the two lines matching the "`love`" pattern.

When you are creating a pattern, you can take advantage of symbols like the asterisk `*` wildcard. Basically `*` means "*anything can match here*". If we use the wildcard in place of the file to search through, we are telling `grep` to search through *any available file*. Let's look for the pattern "see" in all available files in the directory:

```
grep see *
```

Two results! One is from `longReadMe` and the other is from `readMe.txt`. As you can see, `grep` went through every file in the current directory.
What if you want to search through only text files? Use `*.txt`.
What if you want to search through files whose names start with "hw"? Use `hw*`.
What about starting with "hw" and ending in "txt"? Use `hw*.txt`!
The possibilities are endless!

Our current `grep` command only looks through files in the current directory, though. We can extend this to search through all subdirectories as well, using the recursive `-R` option. Now `grep` will search through files in directories like `maze`, and through its subdirectories, and its subdirectories' subdirectories, etc.

```
grep -R agent *
```

Let's break this command down. You are searching recursively through all available files in all directories and subdirectories for the pattern "agent". You'll see that there are many matches in files way inside the `maze` directory.

Pretend you had this awesome program called "`Forgotten.java`". You wrote it last year, but now you forgot where exactly you put it. You just know it's somewhere in `maze`. Ever had this happen to you?

> `find` – searches recursively for files in a directory by name

Naturally, the `find` command will search through all subdirectories and display the name and path of files it encounters. Unless you give it extra criteria of what to look for, it will just print out all encountered files. Let's see that in action:

```
find
```

Whoa, it just printed out names of all files in the starter kit. You'll notice that `maze` has quite a few files. Right now that's way too much information. We need to narrow it down with some search criteria.

First, we know `Forgotten.java` is somewhere in the `maze` directory, so let's focus our search there.

```
find maze
```

Second, we know the name of the file is something along the lines of Forgotten.java, so let's add the `-name` option to find to specify what name to search for:

```
find maze -name Forgotten.java
```

Aha! Found it in `maze/lotsaStuffInHere/common/data/sky`.

Uh-oh, what if this is not the exact "forgotten" file we were looking for? Let's relax our search criteria a bit by using a wildcard in the file name.

```
find maze -name Forgotten*
```

The above command will search `maze` for any file with a name that begins with "Forgotten". As you can see, we find a number of other files such as `Forgotten.java`, `ForgottenForever.java`, and `ForgottenForeverAndEver.java`.

How sad.

COMPLETING/SUBMITTING HOMEWORK

- **Objective:** Learn how to test a basic Java program on Linux machines and submit it to a grader using `turnin`.
- **Commands**: `javac, java, turnin`

Often times one of the criteria for an assignment is that *it must run on the Linux machines*. If the grader tests your program and it fails on the UTCS Linux machines, then that's the end of the story. They don't care that it "ran on your computer".

Let's go through compiling, running and testing a basic Java program. If you don't know about compiling and running programs yet, don't worry you'll learn it in your first class! Pretend the code you are turning in for your first assignment is `HelloWorldBroken.java` (inside the Starter Kit). Now, if you couldn't tell, this program is going to have a bug in it. It will fail to compile and display an error message. Invoke the Java compiler on `HelloWorldBroken.java`:

```
javac HelloWorldBroken.java
```

Oops, you have some naming discrepancies between the class name and file name. Like the good student you are, you would proceed to fix this bug flawlessly. Let's pretend the result would be `HelloWorld.java`. Now let's try our luck compiling our fixed code:

```
javac HelloWorld.java
```

Wait, nothing happened? Well, if there are no error messages then that generally means it succeeded! You should now be able to run your program by using the `java` command with the name of your program (*not the name of your file*):

```
java HelloWorld
```

`javac` – invoke the Java compiler to compile your code

`java` – interpret and run your Java program

Perfecto. Now you know your code compiles and runs on the Linux machines. Let's go through that *one more time* just for laughs:

```
javac HelloWorld.java
java HelloWorld
```

Okay finally, how do you use that pesky homework turnin command? When you've completed your perfect programming assignment and have all the files assembled into one place, you can use `turnin`.

`turnin` – a special homework command, used to submit an assignment to a grader

Note this is **NOT** the same turnin as the online MicroLab Turnin System. The `turnin` we're talking about is entirely Linux-based! Make sure you check the syllabus to see which your class is expected to use (the online system is usually for introductory courses).

You would enter `turnin` commands in this form:

```
turnin --submit <GRADER EID> <ASSIGNMENT NAME>  file(s)
```

You could submit a single file at the end, a list of files one after another, or a directory full of files. Let's say I want to submit my "hw1" assignment to my TA, whose EID is `foo999`. Here might be some examples of what I would submit:

```
turnin --submit foo999 hw1 helloWorld.java
turnin --submit foo999 hw1 part1.java part2.java
turnin --submit foo999 hw1 directoryForHW1
```

Make sure you double-check with your teacher to see exactly what format you should submit your assignments in (i.e., multiple files, a directory, etc.)!

If you're paranoid like me, you'll want to verify that the assignment was successfully submitted and the grader has the files you intended them to grade. You can download your submitted files by using the `--verify` option:

```
turnin --verify <GRADER EID> <ASSIGNMENT NAME>
```

Your assignment files will be downloaded to your current directory, so sometimes it's smart to create a new folder to verify your files in. Create a new directory called `verification`, enter it, and verify the `hw1` assignment by downloading your submitted files:

```
mkdir verification
cd verification
turnin --verify foo999 hw1
```

WHAT TO DO NEXT

That's it for now folks! You should now have the most basic knowledge you need to get started with Linux. But of course, there's still tons more to learn. Look online for more info about Linux commands, files, processes, etc. Buy a book! It's well worth your time to get a handle on it now rather than later.

Recommended topics for further reading: File permissions, pipes, redirects, recursion, command-line editors (such as vim or emacs), aliases
Recommended commands to research: `pushd, popd, sudo, ps, chown, chmod, wc, watch, tee`